Secure Gateways

9 minute read ✓ page test

The Control Ingress Traffic task describes how to configure an ingress gateway to expose an HTTP service to external traffic. This task shows how to expose a secure HTTPS service using either simple or mutual TLS.

Before you begin

1. Perform the steps in the Before you begin. and Determining the ingress IP and ports sections of the Control Ingress Traffic task. After performing those steps you should have Istio and the `httpbin` service deployed, and the environment variables `INGRESS_HOST` and `SECURE_INGRESS_PORT` set.
2. For macOS users, verify that you use `curl` compiled with the LibreSSL library:

```
$ curl --version | grep LibreSSL
curl 7.54.0 (x86_64-apple-darwin17.0) libcurl/7.54.0 LibreS
SL/2.0.20 zlib/1.2.11 nghttp2/1.24.0
```

If the previous command outputs a version of LibreSSL as shown, your `curl` command should work correctly with the instructions in this task. Otherwise, try a different implementation of `curl`, for example on a Linux machine.

Generate client and server certificates and keys

For this task you can use your favorite tool to generate certificates and keys. The commands below use `openssl`

1. Create a root certificate and private key to sign the certificates for your services:

   ```
   $ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:20
   48 -subj '/O=example Inc./CN=example.com' -keyout example.c
   om.key -out example.com.crt
   ```

2. Create a certificate and a private key for `httpbin.example.com`:

```
$ openssl req -out httpbin.example.com.csr -newkey rsa:2048
 -nodes -keyout httpbin.example.com.key -subj "/CN=httpbin.
example.com/O=httpbin organization"
$ openssl x509 -req -days 365 -CA example.com.crt -CAkey ex
ample.com.key -set_serial 0 -in httpbin.example.com.csr -ou
t httpbin.example.com.crt
```

Configure a TLS ingress gateway for a single host

1. Ensure you have deployed the `httpbin` service from Before you begin.

2. Create a secret for the ingress gateway:

```
$ kubectl create -n istio-system secret tls httpbin-credent
ial --key=httpbin.example.com.key --cert=httpbin.example.co
m.crt
```

3. Define a gateway with a `servers:` section for port 443, and specify values for `credentialName` to be `httpbin-credential`. The values are the same as the secret’s name. The TLS mode should have the value of `SIMPLE`.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: mygateway
spec:
  selector:
    istio: ingressgateway # use istio default ingress gateway
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: httpbin-credential # must be the same as secret
    hosts:
    - httpbin.example.com
EOF
```

4. Configure the gateway’s ingress traffic routes. Define the corresponding virtual service.

```
$ cat <<EOF | kubectl apply -f -
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
  - "httpbin.example.com"
  gateways:
  - mygateway
  http:
  - match:
    - uri:
        prefix: /status
    - uri:
        prefix: /delay
    route:
    - destination:
        port:
          number: 8000
        host: httpbin
```

```
EOF
```

5. Send an HTTPS request to access the `httpbin` service through HTTPS:

   ```
   $ curl -v -HHost:httpbin.example.com --resolve "httpbin.example.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
   --cacert example.com.crt "https://httpbin.example.com:$SECURE_INGRESS_PORT/status/418"
   ```

   The `httpbin` service will return the 418 I'm a Teapot code.

6. Delete the gateway’s secret and create a new one to change the ingress gateway’s credentials.

   ```
   $ kubectl -n istio-system delete secret httpbin-credential
   ```

```
$ mkdir new_certificates
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -subj '/O=example Inc./CN=example.com' -keyout new_certificates/example.com.key -out new_certificates/example.com.crt
$ openssl req -out new_certificates/httpbin.example.com.csr -newkey rsa:2048 -nodes -keyout new_certificates/httpbin.example.com.key -subj "/CN=httpbin.example.com/O=httpbin organization"
$ openssl x509 -req -days 365 -CA new_certificates/example.com.crt -CAkey new_certificates/example.com.key -set_serial 0 -in new_certificates/httpbin.example.com.csr -out new_certificates/httpbin.example.com.crt
$ kubectl create -n istio-system secret tls httpbin-credential \
--key=new_certificates/httpbin.example.com.key \
--cert=new_certificates/httpbin.example.com.crt
```

7. Access the `httpbin` service using `curl` using the

new certificate chain:

```
$ curl -v -HHost:httpbin.example.com --resolve "httpbin.example.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
--cacert new_certificates/example.com.crt "https://httpbin.example.com:$SECURE_INGRESS_PORT/status/418"
...
HTTP/2 418
...
    -=[ teapot ]=-


       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
```

8. If you try to access `httpbin` with the previous certificate chain, the attempt now fails.

```
$ curl -v -HHost:httpbin.example.com --resolve "httpbin.example.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
--cacert example.com.crt "https://httpbin.example.com:$SECURE_INGRESS_PORT/status/418"
...
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, Server hello (2):
* curl: (35) error:04FFF06A:rsa routines:CRYPTO_internal:block type is not 01
```

Configure a TLS ingress

gateway for multiple hosts

You can configure an ingress gateway for multiple hosts, `httpbin.example.com` and `helloworld-v1.example.com`, for example. The ingress gateway retrieves unique credentials corresponding to a specific `credentialName`.

1. To restore the credentials for `httpbin`, delete its secret and create it again.

   ```
   $ kubectl -n istio-system delete secret httpbin-credential
   $ kubectl create -n istio-system secret tls httpbin-credent
   ial \
   --key=httpbin.example.com.key \
   --cert=httpbin.example.com.crt
   ```

2. Start the `helloworld-v1` sample

```
$ cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: helloworld-v1
  labels:
    app: helloworld-v1
spec:
  ports:
  - name: http
    port: 5000
  selector:
    app: helloworld-v1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-v1
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld-v1
      version: v1
  template:
    metadata:
      labels:
        app: helloworld-v1
        version: v1
    spec:
      containers:
      - name: helloworld
        image: istio/examples-helloworld-v1
        resources:
          requests:
            cpu: "100m"
        imagePullPolicy: IfNotPresent #Always
        ports:
```

```
        - containerPort: 5000
EOF
```

3. Generate a certificate and a private key for `helloworld-v1.example.com`:

   ```
   $ openssl req -out helloworld-v1.example.com.csr -newkey rs
   a:2048 -nodes -keyout helloworld-v1.example.com.key -subj "
   /CN=helloworld-v1.example.com/O=helloworld organization"
   $ openssl x509 -req -days 365 -CA example.com.crt -CAkey ex
   ample.com.key -set_serial 1 -in helloworld-v1.example.com.c
   sr -out helloworld-v1.example.com.crt
   ```

4. Create the `helloworld-credential` secret:

   ```
   $ kubectl create -n istio-system secret tls helloworld-cred
   ential --key=helloworld-v1.example.com.key --cert=helloworl
   d-v1.example.com.crt
   ```

5. Define a gateway with two server sections for port 443. Set the value of `credentialName` on each port to `httpbin-credential` and `helloworld-credential` respectively. Set TLS mode to `SIMPLE`.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: mygateway
spec:
  selector:
    istio: ingressgateway # use istio default ingress gatew
ay
  servers:
  - port:
      number: 443
      name: https-httpbin
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: httpbin-credential
    hosts:
    - httpbin.example.com
  - port:
      number: 443
      name: https-helloworld
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: helloworld-credential
    hosts:
    - helloworld-v1.example.com
EOF
```

6. Configure the gateway’s traffic routes. Define the corresponding virtual service.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: helloworld-v1
spec:
  hosts:
  - helloworld-v1.example.com
  gateways:
  - mygateway
  http:
  - match:
    - uri:
        exact: /hello
    route:
    - destination:
        host: helloworld-v1
        port:
          number: 5000
EOF
```

7. Send an HTTPS request to `helloworld-v1.example.com`:

   ```
   $ curl -v -HHost:helloworld-v1.example.com --resolve "hello
   world-v1.example.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
   --cacert example.com.crt "https://helloworld-v1.example.com
   :$SECURE_INGRESS_PORT/hello"
   HTTP/2 200
   ```

8. Send an HTTPS request to `httpbin.example.com` and still get a teapot in return:

```
$ curl -v -HHost:httpbin.example.com --resolve "httpbin.exa
mple.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
--cacert example.com.crt "https://httpbin.example.com:$SECU
RE_INGRESS_PORT/status/418"
...
    -=[ teapot ]=-

       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
```

Configure a mutual TLS

ingress gateway

You can extend your gateway’s definition to support mutual TLS. Change the credentials of the ingress gateway by deleting its secret and creating a new one. The server uses the CA certificate to verify its clients, and we must use the name cacert to hold the CA certificate.

```
$ kubectl -n istio-system delete secret httpbin-credential
$ kubectl create -n istio-system secret generic httpbin-credenti
al --from-file=tls.key=httpbin.example.com.key \
--from-file=tls.crt=httpbin.example.com.crt --from-file=ca.crt=e
xample.com.crt
```

1. Change the gateway's definition to set the TLS mode to `MUTUAL`.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: mygateway
spec:
 selector:
   istio: ingressgateway # use istio default ingress gatewa
y
 servers:
 - port:
     number: 443
     name: https
     protocol: HTTPS
   tls:
     mode: MUTUAL
     credentialName: httpbin-credential # must be the same
as secret
   hosts:
   - httpbin.example.com
EOF
```

2. Attempt to send an HTTPS request using the prior approach and see how it fails:

```
$ curl -v -HHost:httpbin.example.com --resolve "httpbin.exa
mple.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
--cacert example.com.crt "https://httpbin.example.com:$SECU
RE_INGRESS_PORT/status/418"
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Request CERT (13):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Certificate (11):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* TLSv1.3 (IN), TLS alert, unknown (628):
* OpenSSL SSL_read: error:1409445C:SSL routines:ssl3_read_b
ytes:tlsv13 alert certificate required, errno 0
```

3. Generate client certificate and private key:

```
$ openssl req -out client.example.com.csr -newkey rsa:2048 -nodes -keyout client.example.com.key -subj "/CN=client.example.com/O=client organization"
$ openssl x509 -req -days 365 -CA example.com.crt -CAkey example.com.key -set_serial 1 -in client.example.com.csr -out client.example.com.crt
```

4. Pass a client certificate and private key to `curl` and resend the request. Pass your client’s certificate with the `--cert` flag and your private key with the `--key` flag to `curl`.

```
$ curl -v -HHost:httpbin.example.com --resolve "httpbin.exa
mple.com:$SECURE_INGRESS_PORT:$INGRESS_HOST" \
--cacert example.com.crt --cert client.example.com.crt --ke
y client.example.com.key \
"https://httpbin.example.com:$SECURE_INGRESS_PORT/status/41
8"
...
    -=[ teapot ]=-

       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
```

More info

Key formats

Istio supports reading a few different Secret formats, to support integration with various tools such as cert-manager:

- A TLS Secret with keys `tls.key` and `tls.crt`, as described above. For mutual TLS, a `ca.crt` key can be used.

- A generic Secret with keys `key` and `cert`. For mutual TLS, a `cacert` key can be used.
- A generic Secret with keys `key` and `cert`. For mutual TLS, a separate generic Secret named `<secret>-cacert`, with a `cacert` key. For example, `httpbin-credential` has `key` and `cert`, and `httpbin-credential-cacert` has `cacert`.
- The `cacert` key value can be a CA bundle consisting of concatenated individual CA certificates.

SNI Routing

An HTTPS `Gateway` with a `hosts` field value other than `*` will perform SNI matching before forwarding a request, which may cause some requests to fail. See configuring SNI routing for details.

Troubleshooting

- Inspect the values of the `INGRESS_HOST` and `SECURE_INGRESS_PORT` environment variables. Make sure they have valid values, according to the output of the following commands:

```
$ kubectl get svc -n istio-system
$ echo "INGRESS_HOST=$INGRESS_HOST, SECURE_INGRESS_PORT=$SE
CURE_INGRESS_PORT"
```

- Check the log of the `istio-ingressgateway` controller for error messages:

```
$ kubectl logs -n istio-system "$(kubectl get pod -l istio=
ingressgateway \
-n istio-system -o jsonpath='{.items[0].metadata.name}')"
```

- If using macOS, verify you are using `curl` compiled with the LibreSSL library, as described in the Before you begin section.
- Verify that the secrets are successfully created in

the istio-system namespace:

```
$ kubectl -n istio-system get secrets
```

httpbin-credential and helloworld-credential should show in the secrets list.

- Check the logs to verify that the ingress gateway agent has pushed the key/certificate pair to the ingress gateway.

```
$ kubectl logs -n istio-system "$(kubectl get pod -l istio=
ingressgateway \
-n istio-system -o jsonpath='{.items[0].metadata.name}')"
```

The log should show that the httpbin-credential secret was added. If using mutual TLS, then the `httpbin-credential-cacert` secret should also appear. Verify the log shows that the gateway agent receives SDS requests from the ingress gateway, that the resource's name is `httpbin-credential`, and that the ingress gateway obtained the key/certificate pair. If using mutual TLS, the log should show key/certificate was sent to the ingress gateway, that the gateway agent received the SDS request with the `httpbin-credential-cacert` resource name, and that the ingress gateway obtained the root certificate.

Cleanup

1. Delete the gateway configuration, the virtual service definition, and the secrets:

    ```
    $ kubectl delete gateway mygateway
    $ kubectl delete virtualservice httpbin
    $ kubectl delete --ignore-not-found=true -n istio-system se
    cret httpbin-credential \
    helloworld-credential
    $ kubectl delete --ignore-not-found=true virtualservice hel
    loworld-v1
    ```

2. Delete the certificates and keys:

```
$ rm -rf example.com.crt example.com.key httpbin.example.com.crt httpbin.example.com.key httpbin.example.com.csr helloworld-v1.example.com.crt helloworld-v1.example.com.key helloworld-v1.example.com.csr client.example.com.crt client.example.com.csr client.example.com.key ./new_certificates
```

3. Shutdown the `httpbin` and `helloworld-v1` services:

```
$ kubectl delete deployment --ignore-not-found=true httpbin helloworld-v1
$ kubectl delete service --ignore-not-found=true httpbin helloworld-v1
```